Computational Statistics manuscript No.
(will be inserted by the editor)

Eulerian tour algorithms for data visualization and the `PairViz` package

C.B. Hurley* **and R.W. Oldford****

Received: date / Accepted: date

Abstract `PairViz` is an R package that produces orderings of statistical objects for visualization purposes. We abstract the ordering problem to one of constructing edge-traversals of (possibly weighted) graphs. `PairViz` implements various edge traversal algorithms which are based on Eulerian tours and Hamiltonian decompositions. We describe these algorithms, their `PairViz` implementation and discuss their properties and performance. We illustrate their application to two visualization problems, that of assessing rater agreement, and model comparison in regression.

Keywords Hamiltonian · Eulerian tour · seriation · visualization · parallel coordinates

1 Introduction

Visualization methods are important in the exploration, analysis and presentation of data. A carefully chosen graphic creates a visual impression of the overall behaviour of a dataset or a model fit. At a more detailed level, comparisons become important, for example, comparisons between variables, cases, groups, clusters or models. A common practice is to lay out the objects in a line and compare them to one another on a common aligned scale. However, as the visual distance between the objects being compared increases, the accuracy of the pairwise comparison decreases. This simple linear layout unwittingly favours comparisons between adjacent objects, that is, $n-1$ of the possible $\binom{n}{2}$ pairwise comparisons between n objects.

In the visualization literature, a number of approaches have been taken to finding informative linear orderings. These include (i) interactively picking and dropping objects to facilitate comparison (for example Theus 2002; Yang et al. 2003), (ii) sorting the objects on some characteristic (e.g. Cleveland 1995, Theus 2002 and Hofmann 2006) and (iii) seriating the objects so that nearby objects are similar (for example, Ankerst et al. 1998; Friendly and Kwan 2003) or more generally, where their comparison is "interesting" (Hurley 2004).

In Hurley and Oldford (2010a) we took a different though complementary approach to the visualization of comparisons. We constructed visualizations which facilitated all $\binom{n}{2}$ comparisons, by constructing sequences where all pairs of objects appear adjacently. We also described a number of applications, ranging from parallel coordinate displays to star glyphs and multiple comparisons, and our methodology, which is based on graphs

* Research supported in part by a Research Frontiers Grant from Science Foundation Ireland.

** Research supported in part by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada.

C.B. Hurley
Department of Mathematics, National University of Ireland, Maynooth, Maynooth Co. Kildare, Ireland
Tel.: +353-1-7083792
Fax: +353-1-7083913
E-mail: catherine.hurley@nuim.ie

R.W. Oldford
Department of Statistics and Actuarial Science, University of Waterloo, Waterloo, Ontario, Canada
E-mail: rwoldford@uwaterloo.ca

and graph traversal. All of these ideas are implemented in the R package PairViz (Hurley and Oldford 2009), which is the focus of the present paper.

The PairViz package offers functionality for constructing comparison sequences, and some new varieties of graphics besides. In this paper, we focus on comparison sequences. However, all visualizations presented in the present paper and in Hurley and Oldford (2010a) are provided by PairViz.

In Section 2 we begin with an introductory example where the goal is to compare and relate variables. Section 3 gives a synopsis of the relevant graph theory – Hurley and Oldford (2010a) gives more details. Section 4 describes and gives examples of PairViz functions for constructing comparison sequences, which are actually graph traversals. The following Section 5 compares these functions, and offers guidance in assessing which algorithm is appropriate for the visualization task at hand. Section 6 explores the use of graph traversal in model selection for regression, and finally we give concluding remarks in Section 7.

2 An introductory example

We start with a motivating example. The "diagnoses" data (Fleiss 1971), taken here from the R package irr of Gamer et al (2009), contains psychiatric diagnoses of 30 patients provided by 6 raters. The ratings are D=depression, PD= personality disorder, S= Schizophrenia, N= Neurosis, O= Other. Figure 1 is a graphical

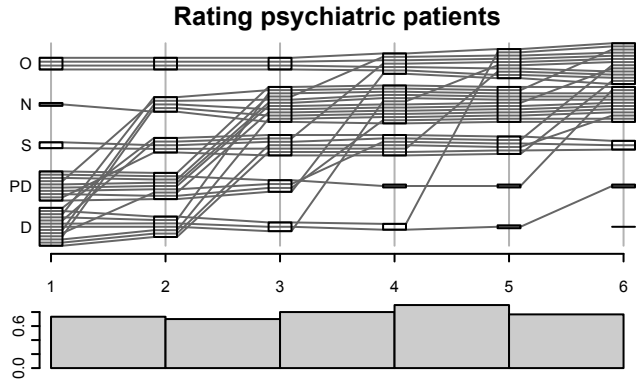


Fig. 1 Parallel coordinate plot of ratings, one axis per rater. Overlaid rectangles show each rater's distribution of ratings.The lower bar chart shows the agreement proportion for each adjacent pair of raters.

comparison of the raters using a parallel coordinate display, adapted for categorical data. There is one parallel axis for each of the six raters. The ratings are assigned nominal values of "1" for depression, "2" for personality disorder and so on. Then, for each rater, coordinates are spread out vertically in an equispaced fashion about the nominal value. Also, rather than 0-width axes as is usual in parallel coordinate displays, the axes are given a small width so that on each axis, observations are represented by horizontal line segments. The overlaid rectangles have length proportional to frequency and so give the marginal distribution for each rater. With this representation we can compare raters, and also the ratings given to a patient by different raters. We see for instance that rater 6 never assigns a rating of "Depression", while this diagnosis is popular with raters 1 and 2.

This adapted parallel coordinate display is similar to *parallel sets*, as proposed by Bendix et al. (2005). Unwin et al. (2003) alternatively overlaid circles with area proportional to frequency on the parallel axes to show marginal distributions of categorical variables, while Wills (2000) overlaid marginal summary views on axes for both categorical and continuous variables.

One problem with the visualization of Figure 1 is that it is clearly easier to assess agreement between raters who appear on adjacent axes. For example, the barchart shows that each pair of consecutive raters has agreement of 0.5 or more, but it is difficult to assess agreement between other pairs. The main theme of our research and the PairViz package is that visualizations should facilitate all interesting comparisons. For the diagnoses data, this is achieved with a parallel coordinate display where all pairs of raters appear adjacently, as shown in Figure 2.

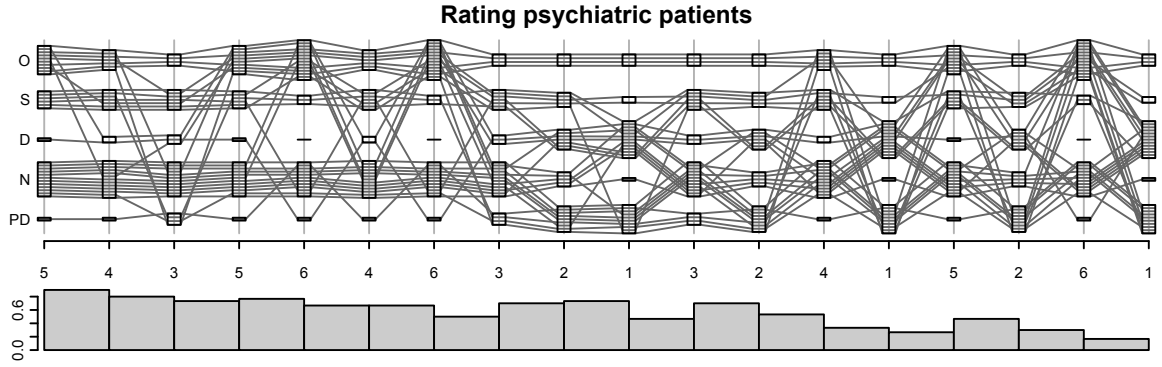


Fig. 2 Parallel coordinate plot of ratings. Each axis represents a rater, all pairs of raters appear adjacently. Overlaid rectangles show each rater's distribution of ratings. The lower bar chart shows the agreement proportion for each adjacent pair of raters.

The ordering for the horizontal axis is produced using the `PairViz` function `eulerian`, which will be described in Section 4.4. But the main idea should be clear: all pairs of raters appear adjacently at least once, and the pairs are ordered in such a way that the raters whose agreement is higher tend to appear first. To verify that this is the case, the barchart in the lower panel of Figure 2 traces the proportion of ratings that are in agreement between each pair of raters. This display shows that raters 4 and 5 have the highest agreement, and that each of these also agrees well with raters 3 and 6. Rater 1 has low agreement with all raters except rater 2, while rater 2 has good agreement with raters 1 and 3 only. Thus Figure 1 could give a misleading impression of the level of agreement between raters.

The observant reader may have noticed that in Figure 2 we have used different nominal values and thus different orderings of the diagnoses on the vertical axis. This re-ordering was chosen so frequently confused diagnoses are placed adjacently. With this re-ordering, the parallel coordinate plot line segments are shorter and consequently less tangled and easier to follow. The reader may also have noticed that the pairs of raters (2,3) and (4,6) appear adjacently twice. The reason for this will be clear on reading the next section.

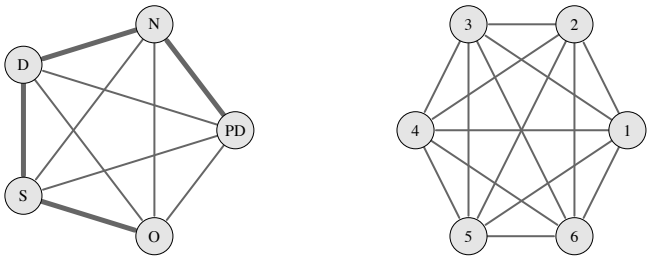
Finally, we remark that the data as analysed above is artificial. Checking the description of the data as given by Fleiss (1971), there were a total of 43 psychiatrists, six of whom rated each patient. In the analysis above, we assumed there were 6 psychiatrists each rating all 30 patients. In the data listed in the `irr` package, the six ratings on a patient are sorted by their nominal values. This explains the upward trend of ratings and the apparent agreement between adjacent raters in Figure 1 .

3 Brief Background

In this section, we review some classical graph theory, and refer the reader to Gross and Yellen (2004) and Hurley and Oldford (2010a, 2010b) for more details. Our setup is as follows. The objects we want to order are identified with nodes (vertices) of a graph. Undirected edges are placed between nodes which we wish to compare. Edge weights indicate the importance of the comparison, the smaller the edge weight the more important the comparison. In many applications we are interested in all comparisons, in which case the graph is said to be *complete*. Such a graph is denoted by K_n where n is the number of nodes. Generally, we are interested in traversing the graph in such a way as to visit every edge.

A path is called a *Hamiltonian path* if it visits all vertices of a graph exactly once. A path which contains all of the edges of a graph, visiting each edge exactly once is called an *Eulerian path* (or *Eulerian trail)* and if the path is closed then the traversal is called an *Eulerian tour*.

For example, Figure 3 shows the two graphs used in the construction of Figure 2. The first has the diagnoses as nodes, and is complete. The path shown in dark grey is (one of many) open Hamiltonian paths on this graph. Here we use an edge weight which is small if the two diagnoses are frequently assigned to the same patient, and the path shown is the open Hamiltonian path for which the total of edge weights is smallest. This path gives the ordering of diagnoses used in Figure 2. The graph of Figure 3(b) is also a complete graph, with the raters as nodes. In this case we use an edge weight which is the proportion of patients for which



(a) Hamiltonian on diagnoses graph (b) Complete graph of raters

Fig. 3 Graphs used in the construction of Figure 2.

two raters disagree. The ordering of raters used in Figure 2 visits every edge of the Figure 3(b) graph at least once, starting with low-weight edges.

It is well known that if G is a connected graph, G has an Eulerian tour if and only if it is an even graph (i.e. every vertex has an even number of edges). (This result goes back to Euler in 1763, and his solution to the Bridges of Konigsberg problem). Furthermore, a connected graph G with exactly two odd nodes has an open Eulerian trail starting and ending at the odd nodes. The graph K_{2m+1} is an even graph, and thus has an Eulerian tour. (In fact, as discussed in Hurley and Oldford (2010a), it has an immense number of such tours). Furthermore, it is possible to construct Eulerian tours on K_{2m+1} which are *Hamiltonian decompositions*, that is, composed entirely of edge-distinct Hamiltonian cycles.

The graph K_{2m} is not an even graph, in fact all of its vertices have an odd number of edges. It follows that K_{2m} does not have an Eulerian path. Any path that visits all of the edges will have to visit some edges more than once. Our constructions produce edge-traversals on K_{2m} which are "nearly Eulerian" by adding $m-1$ extra edges to K_{2m} in such a way that $2m-2$ of the nodes are even and two are odd. We denote this extended version of K_{2m} by K^e_{2m}. Then an open Eulerian path exists on K^e_{2m}, starting at one odd node and ending at the other, visiting all edges along the way.

For example, Figure 3(b) shows K_6, which is not Eulerian. However the path visited by the sequence of axes of Figure 2 is an Eulerian on K^e_6, formed by adding duplicate edges between nodes (2,3) and (4,6). Thus the path begins at one odd node (5), and ends at the other (1), and visits every edge of K^e_6.

Since K_{2m} is not Eulerian it does not have a decomposition into edge-distinct Hamiltonian cycles. However, it is possible to decompose K_{2m} into edge-distinct Hamiltonian *paths*, rather than cycles. Here are some general results about Hamiltonian cycle or path decompositions on the complete graph. K_n can be decomposed as follows:

(lw-odd-cycle) For $n = 2m+1$, into m Hamiltonian cycles, or
(lw-odd-path) m Hamiltonian paths and an almost-one factor.
(lw-even-path) For $n = 2m$, into either m Hamiltonian paths, or
(lw-even-cycle) $m-1$ Hamiltonian cycles and a one-factor (or perfect matching).

These are known as Lucas-Walecki decompositions (Lucas 1892; also Alspach et al 1990). See also Hurley and Oldford (2010a, 2010b).

In the following sections, we will refer to constructing Eulerians on K_n. Strictly speaking, for even $n = 2m$ the path is only a nearly Eulerian path on K_{2m} but is a proper open Eulerian path on K^e_{2m}.

4 Constructing Eulerians

Eulerian paths by definition must visit every edge of a graph. However, in visualization contexts the order in which edges are visited is important, as viewers can generally absorb more information more easily when that information is presented in an ordered and organized fashion.

For instance, edge-weighted graphs recognize the fact that some edges are more important than others. In Figure 3(b), we use an edge weight which is the proportion of patients for which the two raters disagree. The Eulerian path selected for the visualization of Figure 2 starts from the lowest weight edge, that is, the

two raters with the lowest disagreement or equivalently the highest agreement. Thereafter the path continues to follow low-weight edges.

In this section, we describe the four functions in the `PairViz` package for constructing Eulerians. Each of these functions produces Eulerians with different properties. The first, `hpaths`, produces Hamiltonian decompositions on complete graphs. The second method, `weighted_hpaths`, produces Hamiltonian decompositions on weighted complete graphs, where the Hamiltonians are roughly weight increasing. The third method is a recursive algorithm which produces Eulerians (not based on Hamiltonians) on complete graphs, and this is implemented in the function `eseq`. Finally, we describe the function `etour`, for producing weight-increasing Eulerians on weighted graphs. This is the function used to construct the sequence of Figure 2.

4.1 Hamiltonian decompositions on complete graphs

Lucas-Walecki constructions provide a method for constructing all four varieties of decomposition on K_n described in the previous section. In Hurley and Oldford (2010a) we described constructions for the decompositions lw-odd-cycle and lw-even-path only. However, the `PairViz` package provides all four decompositions, i.e., lw-odd-cycle, lw-odd-path, lw-even path and lw-even-cycle.

Our basic work horse is the `zigzag` function. This function yields an $n \times m$ matrix with $m = \lceil n/2 \rceil$, where each row is a Hamiltonian on K_n. For example, see the following results:

```
> zigzag(6)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    6    3    5    4
[2,]    2    3    1    4    6    5
[3,]    3    4    2    5    1    6
```

```
> zigzag(7)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    2    7    3    6    4    5
[2,]    2    3    1    4    7    5    6
[3,]    3    4    2    5    1    6    7
[4,]    4    5    3    6    2    7    1
```

Note that when the n vertices are arranged clockwise around a circle, each row follows a zig-zag path starting with node i, for $i = 1,2,\ldots,m$, In general `zigzag(2m)` produces the decomposition lw-even-path, and `zigzag(2m+1)` yields lw-odd-path. Notice for example that the last row of `zigzag(7)` duplicates edges (4,5), (3,6) and (2,7) from the first row. The other edges (5,3), (6,2) and (7,1) from the last row are the almost-one factor.

We mention one early application of the zig-zag construction in statistical graphics. Wegman (1990) produced $m = \lceil n/2 \rceil$ different parallel coordinate displays of $n-$variable data, where the ith display used the variable permutation given by the ith row of `zigzag(n)`.

Next we consider the Hamiltonian cycle decompositions, namely, lw-odd-cycle and lw-even-cycle. First, notice `zigzag(6)` has each of $1,\ldots,6$ as endpoints of the rows. Therefore, binding a column of 7's to `zigzag(6)` produces a Hamiltonian cycle decomposition on K_7. Similarly, binding a column of 8's to `zigzag(7)` yields three edge-disjoint Hamiltonian cycles in the first three rows, with the last row containing a one-factor (perfect matching). In general, `cbind(n, zigzag(n-1))` produces the cycle decompositions lw-odd-cycle and lw-even-cycle. In our implementation, however, we use instead `cbind(1, 1+ zigzag(n-1))` so that all four decompositions start with 1.

The `hpaths` function produces each of the four decompositions detailed above. Cycles or paths are requested via the argument `cycle`. The default behaviour is for `hpaths` to produce the exact decomposition, i.e. cycles for odd n and paths for even n. For example, `hpaths(7)` produces decomposition lw-odd-cycle:

```
> hpaths(7)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    2    3    7    4    6    5
[2,]    1    3    4    2    5    7    6
[3,]    1    4    5    3    6    2    7
```

When the rows of `hpaths(7)` are glued together and a final 1 added to close the path, we have an Eulerian tour on K_7. Invoking `hpaths` with the `matrix=FALSE` option produces the tour.

```
> hpaths(7,matrix=FALSE)
 [1] 1 2 3 7 4 6 5 1 3 4 2 5 7 6 1 4 5 3 6 2 7 1
```

As described above, `hpaths(6)` produces the decomposition lw-even-path as given by `zigzag(6)`, and when the paths, that is the rows, are joined together the joining edges are now visited twice. In fact `hpaths(6,matrix=FALSE)` gives an Eulerian path not on K_6 but on the amended graph K_6^e. In general the duplicated edges of this construction are $(j, j+m-1)$, $j = 2,\ldots,m$ for the graph K_{2m}.

Figure 4 plots the sequences produced by `hpaths(8)` and `hpaths(9)`. The black edges on `hpaths(8)`

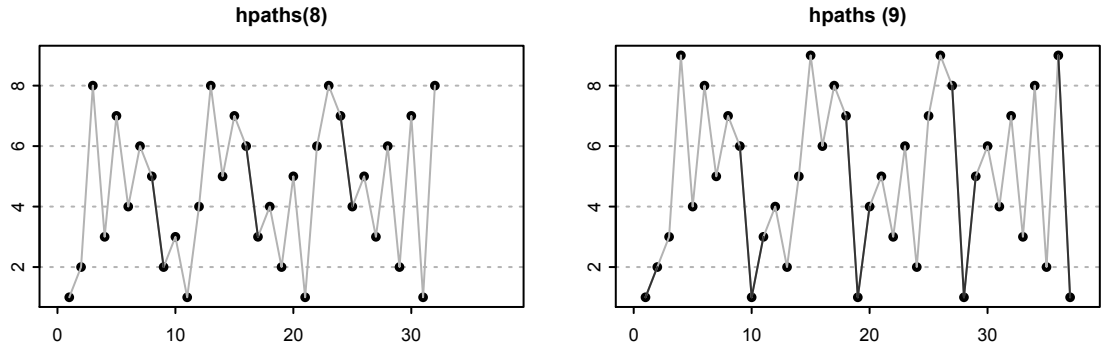


Fig. 4 Traces of the Eulerian sequences on K_8^e and K_9 generated by the function `hpaths`, that is, `hpaths(8)`$[i]$ versus i and `hpaths(9)`$[i]$ versus i,for $i = 1,2,\ldots,N$, where N is the length of the `hpaths` sequence.

connect the four Hamiltonians and these edges appear again elsewhere in the sequence. As described previously, `hpaths(9)` is produced from `hpaths(8)` by adding 1 to every element, breaking the black edges by inserting "1", and appending "1" at the beginning and end. These additional edges are shown in black in the `hpaths(9)` plot of Figure 4.

Other isomorphic decompositions are obtained by supplying the first Hamiltonian as an argument to `hpaths`. Then all node labels are rearranged accordingly, as in the following example:

```
> hpaths(1:7)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    2    3    4    5    6    7
[2,]    1    3    5    2    7    4    6
[3,]    1    5    7    3    6    2    4
```

4.2 Hamiltonian decompositions on weighted complete graphs

For weighted graphs our goal is an ordered Eulerian T where weights tend to increase as the sequence progresses. In Hurley and Oldford (2010a) we proposed an algorithm (referred to as WHam) that builds such paths out of Hamiltonians. In `PairViz` the WHam algorithm is implemented by the function `weighted_hpaths`. This function takes an array of Hamiltonians `h` of size $n \times m$ say, such as that produced by `hpaths`. The `weighted_hpaths` function attempts to pick the best of various rearrangements of this array. We describe the procedure for a cycle decomposition, the modifications for path decompositions are given in parentheses.

1. Find the Hamiltonian cycle (path) with the smallest total weight. (Of course, finding the overall best Hamiltonian is NP hard, so approximate solutions are generally used). The best direction and starting point of the cycle (direction only for paths) is obtained by using a correlation measure on the weights to measure their tendency to increase.

2. The optimized Hamiltonian is placed in the first row of `h`, and node labels in other rows are permuted using this relabeling.
3. Using correlation (as in (1) measuring the tendency of cycles (path) weights to increase), the best direction is chosen for the cycles (paths) in rows $2,\ldots,m$ of `h`.
4. Finally, the cycles (paths) are sorted in order of increasing total weight.

An example of the use of `weighted_hpaths` will be given in Section 5.

4.3 Eulerians on complete graphs, a recursive algorithm

Next we describe an algorithm `eseq` which recursively builds up an Eulerian on K_n. This algorithm has some interesting properties, visiting edges in an "increasing" order.

Suppose we can construct an Eulerian on K_{n-2}. Consider the graph G_n with vertex set $V = \{i, i = 1,2,\ldots n\}$ and edges $(n-1,n)$, $(j,n-1)$ and (j,n) for $j = 1,2,\ldots,n-2$. This graph is illustrated in Figure 5 for the $n = 7$ and $n = 6$ cases. Note that $K_n = K_{n-2} \cup G_n$. Therefore an Eulerian on K_n must visit all of

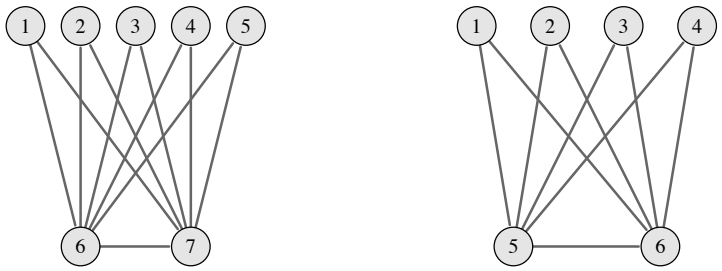


Fig. 5 A graph G_n with edges $(n-1,n)$ and $(j,n-1)$, (j,n) for $j = 1,2,\ldots,n-2$ illustrated for $n = 7$ and $n = 6$. Append Eulerians on K_{n-2} and G_n to form a Eulerian on K_n.

the edges of the graph G_n in addition to those visited by the Eulerian on K_{n-2}. The `eseq` construction forms an Eulerian on K_n by appending Eulerians on K_{n-2} and G_n.

To see how this works, first consider the n odd setting. An Eulerian tour on K_1 is trivially `eseq(1) = 1`. Suppose now that `eseq(n-2)` gives an Eulerian tour on K_{n-2}, starting and ending with 1. Here n is odd and so the graph G_n is even, as illustrated in first graph shown in Figure 5. We form an Eulerian tour T_{new} on G_n by interleaving $n-1$ and n consecutively between elements of the sequence $1,2,\ldots,n-2$, and then appending $n-1,n,1$. That is, $T_{new} = 1,n-1,2,n,3,n-1,4,n,5,\ldots,n-3,n,n-2,n-1,n,1$. This path is joined on to `eseq(n-2)`, resulting in an Eulerian tour on K_n. Note that since the last vertex visited by `eseq(n-2)` coincides with the first vertex of T_{new}, this leading vertex of T_{new} is not repeated when the paths are joined. For example, here are `eseq(5)` and `eseq(7)`.

```
> eseq(5)
[1] 1 2 3 1 4 2 5 3 4 5 1
> eseq(7)
[1] 1 2 3 1 4 2 5 3 4 5 1 6 2 7 3 6 4 7 5 6 7 1
```

For the n even case, the process is initiated with an open Eulerian path on K_2, that is, `eseq(2) = 1,2`. Now suppose we have `eseq(n-2)` which is an Eulerian on K_{n-2} starting with 1 and ending with $n-2$. Here n is even and so the graph G_n has exactly two odd nodes, namely $n-1$ and n, as illustrated in the right hand side graph shown in Figure 5. We form an Eulerian path T_{new} on G_n with the odd nodes as end points, by starting with $n-1$ and then interleaving n and $n-1$ consecutively between elements of the sequence $1,2,\ldots,n-2$, and then appending $n-1,n$. That is, append $T_{new} = n-1,1,n,2,n-1,3,n,4,\ldots,n-3,n,n-2,n-1,n$, onto `eseq(n-2)` to produce `eseq(n)`. Since the last vertex of `eseq(n-2)` and the first of T_{new} differ, the join

implicitly adds another edge, namely $(n-2,n-1)$, and this edge is visited again in T_{new}. Therefore, strictly speaking `eseq(n)` forms an open Eulerian path on K_n^e, where K_n^e is formed by adding additional edges between $(j,j+1)$, for $j = 2,4,6,\ldots,n-2$ to the complete graph K_n. Here we illustrate the results for `eseq(4)` and `eseq(6)`.

```
> eseq(4)
[1] 1 2 3 1 4 2 3 4
> eseq(6)
[1] 1 2 3 1 4 2 3 4 5 1 6 2 5 3 6 4 5 6
```

From its construction, we can see that the Eulerian produced by `eseq(n)` has the following two properties. Since the constructions are slightly different for the cases of odd and even n, we refer to them as `eseq-odd` and `eseq-even` in the following paragraphs. Proofs are provided in the Appendix.

Property 1 For odd n, `eseq-odd(n)` has length $E_n = \binom{n}{2} + 1$. The first E_k elements gives `eseq-odd(k)`, for $k = 1,3,5,\ldots,n-2$.

For even n, `eseq-even(n)` has length $E_n = n^2/2$. The first E_k elements gives `eseq-even(k)`, for $k = 2,4,\ldots,n-2$.

Property 2 Consider two nodes $i < j$, and an integer $k \geq 1$. For most values of i,j,k, the `eseq-odd` and `eseq-even` sequences vist the edge $(i-k,j)$ before (i,j) and the visit to (i,j) precedes a visit to $(i,j+k)$. The exceptions are given in the proof.

We mention two other possible constructions for the n even case. First, start with any Eulerian on K_{n+1} and delete all instances of $n+1$. This creates a closed path on K_n with $n/2$ of the edges visited twice. Deleting one of these duplicated edges provides an open Eulerian path on K_n^e.

The second construction starts with any Eulerian on K_{n-1} and simply inserts at a visit to node '1' the detour $1,n,2,3,n,4,5,\ldots,n-2,n-1,n$. This detour is an open Eulerian path on the graph H_n with vertex set $V = \{i, i = 1,2,\ldots n\}$ and edges (j,n), $j = 1,2,\ldots,n-1$, and $(j,j+1)$, $j = 2,4,\ldots,n-2$. (Of course any Eulerian on H_n would do just as well). The nice feature of this second construction is that when the detour is inserted at the end, duplicated edges appear only in the last section of the Eulerian. These strategies are implemented in `kntour_drop` and `kntour_add`. Note that when these functions are applied to the results of `eseq`, they give the same result. Below we see the result of `kntour_add(eseq(5))` which is identical to `kntour_drop(eseq(7))`.

```
> eseq(5)
 [1] 1 2 3 1 4 2 5 3 4 5 1
> kntour_add(eseq(5))
 [1] 1 2 3 1 4 2 5 3 4 5 1 6 2 3 6 4 5 6
```

Note that if the detour is inserted into the middle of an Eulerian on K_{n-1}, the situation is slightly more complicated. Consider inserting 16236456 at the second occurrence of '1' in `eseq(5)`. The path becomes 1231<u>6236456</u> 14253451 where the detour is underlined. Since the detour does not return to '1' the path breaks between 6 and 1 and is reported as 1425345112316236456.

4.4 Eulerians on general weighted graphs

A classical algorithm for constructing Eulerian tours or open Eulerian paths is due to Hierholzer(1873). In Hurley and Oldford (2010a) we described this algorithm and also a modification (referred to as algorithm GrEul) for weighted graphs. Here we discuss the `PairViz` implementation.

The standard algorithm for Eulerian tours on even graphs builds up the path by starting from an arbitrarily chosen node, and following unvisited edges until no further moves are possible, giving a closed path T. Then starting from a node with unvisited edges, continue to follow unvisited edges, yielding a cycle which is spliced into T. This step is repeated until all edges are visited once. With the GrEul algorithm (Hurley and Oldford

2010a) for weighted graphs, the lowest weight available edge is chosen at each stage. The starting node is also chosen from one of the two nodes adjacent to the lowest weight edge.

In PairViz, the function etour implements the Hierholzer algorithm and its counterpart (GrEul) for weighted graphs. For this we need a data structure which represents a graph. We use the graphNEL class from the R package graph (Gentleman et al. 2010). (As an aside, there are now a number of implementations of graphs available for R. In fact, we used the igraph package (Csardi and Nepusz 2006) to produce all graph drawings in this paper.)

Figure 6 and the adjacent code illustrates the construction of a weighted graph.

```
> n <- LETTERS[1:5]
> g <- new("graphNEL",nodes=n)
> g <- addEdge(n[1],n[2:3],g,8:9)
> g <- addEdge(n[2],n[3:5],g,5:7)
> g <- addEdge(n[4],n[5],g,1)
> etour(g,weighted=FALSE)
[1] "A" "B" "D" "E" "B" "C" "A"
> etour(g)
[1] "E" "D" "B" "C" "A" "B" "E"
```

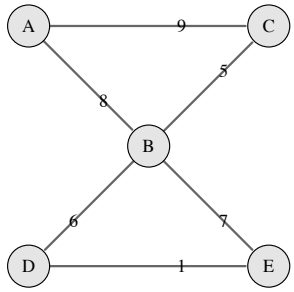


Fig. 6 A weighted graph which is even.

When etour is invoked with weighted=FALSE, the tour starts at the first node of the graph, and always visits the first available edge. Therefore nodes are visited in order ABCA, reaching a dead end at the second visit to node A. The algorithm then restarts from B, the last visited node with unused edges, and visits BDEB, and this path replaces the B in ABCA, giving ABDEBCA.

With weighted=TRUE (the default), the tour starts with edge ED, which is the lowest weight edge in the graph. Here E is chosen over D as the starting node because the next lowest weight edge emanating from D has lower weight than the next lowest weight edge emanating from E. The algorithm follows the lowest weight edge at each stage, giving the path EDBCABE. In this case, all edges are visited without reaching a dead end.

For the construction of complete graphs, the PairViz package provides the generic function mk_complete_graph, which builds a complete graph from a distance matrix (a dist or a symmetric matrix), or with a specified number of vertices. The following demonstrates the result of etour on K_5.

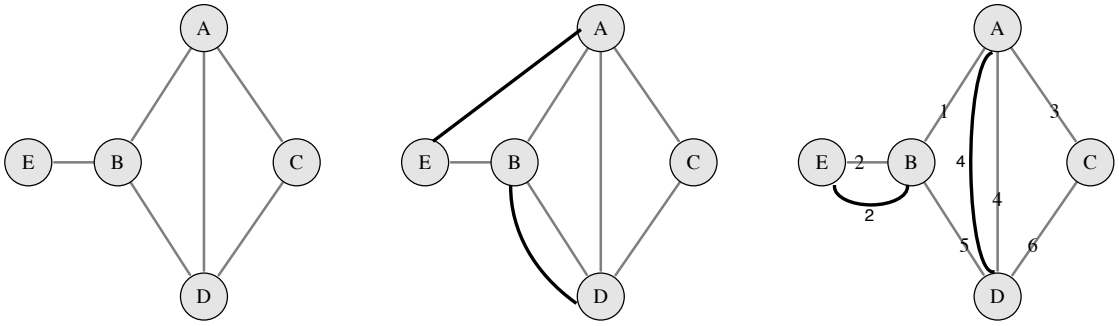
```
> etour(mk_complete_graph(5))
 [1] "1" "2" "3" "1" "4" "2" "5" "3" "4" "5" "1"
```

In the above example, the nodes are ordered numerically, so the algorithm always visits the lowest numbered node available.

Now we shift our focus to graphs where all nodes are not even. For graphs with some odd nodes, it is always possible to form a similar, extended graph which is even. It is a well-known result from graph theory that the number of odd nodes of a graph is even. Using this fact, it is clear that adding additional dummy edges between pairs of odd nodes will produce an even graph. An alternative option would be to augment the graph by adding an extra dummy node with edges to each of the odd nodes, but Eulerian tours from such a graph would include this dummy node.

PairViz provides a function mk_even_graph which constructs an even graph from a graphNEL of a connected graph (or a distance matrix, or with a specified number of vertices), by adding extra edges between pairs of odd nodes. Figure 7 illustrates the process. In this example, the graph of Figure 7(a) has five nodes only one of which (C) has an even number of edges. In Figure 7(b) odd nodes are paired off, A with E and B with D and new edges are added. In the resulting graph, all nodes are now even.

An Eulerian tour on the graph of Figure 7(b) starting from A also ends with A. However, as the edge (E,A) is the last edge visited by the tour, the etour algorithm recognizes that this edge is one of those added by mk_even_graph, and as such does not require a visit. The tour is broken at this edge so the result is actually



(a) The original graph g (b) Graph gb: add two edges to g (c) Graph gc: add two edges to g

Fig. 7 In (a) C is the only even node in g. In (b), extra dummy edges (shown in the thick black line) are (A,E) and (B,D), so the graph gb is even. In (c), the graph gc is weighted and even, with edges are (A,D) and (B,E) added to g.

an open Eulerian path. Here is the result of `etour` on graph gb of Figure 5(b). Note assuming nodes are in lexicographical order, the first node (here node A) is the default starting point for the Eulerian.

```
> etour(gb)
[1] "A" "B" "D" "A" "C" "D" "B" "E"
```

More precisely, the result of `etour(gb)` is an open Eulerian path on the graph with the dummy edge (A,E) deleted from gb. While Eulerian tours exist for even graphs only, open Eulerian paths exist for graphs with exactly two odd nodes. These open paths travel from one odd node to the other, visiting all edges along the way. The `etour` algorithm takes advantage of this fact. When the start node has a dummy edge, say (start, target), this edge is ignored by the edge traversal algorithm. The result is an open Eulerian path, travelling from the start node to the target node.

For the graph of Figure 7(a) the four odd nodes can be paired in three different ways. Figure 7(b) illustrates our pairing strategy for unweighted graphs, that is, pair the first odd node with the last, and the other odd nodes in consecutive pairs. For complete graphs where the number of nodes n is even, this pairing strategy adds edges (1,n) and $(j, j+1)$ for $j = 2,4,\ldots,n-2$. Then the default `etour` path starts at node 1 and ends at node n.

```
> etour(mk_even_graph(6))
 [1] "1" "2" "3" "1" "4" "2" "3" "4" "5" "1" "6" "2" "5" "3" "6" "4" "5" "6"
```

When the graph is weighted, the goal is Eulerian paths with edge weights tending to increase. So the pairing process used by `mk_even_graph` assists in this goal by pairing the default start node with the odd node with the highest average weight, which becomes the target node of the Eulerian path. Figure 7(c) illustrates the pairing process for a weighted graph. Here node A is the default starting point for the tour as it is attached to the lowest weight edge (node A is preferred over node B because the next lowest weight edge emanating from B has lower weight than the next lowest weight edge emanating from A). Our strategy pairs A with the odd node with the highest average weight, which is node D in this instance. This leaves nodes B and E as the other pair. Here we have the results of `etour` on the graph gc of Figure 5(c).

```
> etour(gc)
[1] "A" "B" "E" "B" "D" "A" "C" "D"
> etour(gc,start="C")
[1] "C" "A" "B" "E" "B" "D" "A" "D" "C"
```

When no starting point is specified, A is the default start, which the pairing process partners with node D. Starting from A, `etour` forms an open path ending at D. However when C is specified as a starting node, none of its edges are duplicates, so `etour` forms a tour beginning and ending at C.

As described above, the function `etour` produces an Eulerian on even graphs. The generic function `eulerian` provides a useful wrapper: it builds an `even_graph` (actually a subclass of `graphNEL`) from a

graphNEL, distance matrix, or from the complete graph with a specified number of nodes, and invokes etour on the even_graph created.

The strategy of pairing off the odd nodes in order to construct an Eulerian tour is well-known in graph theory, and arises in the so-called Chinese postman problem (Edmonds and Johnson 1973). The Chinese postman problem is to find the shortest tour that visits every edge at least once. For the graph of Figure 7(a), with weights as in (c), an Eulerian tour of the graph shown in (c) is the solution. Other pairings of the odd nodes would not be allowed, as they add new edges rather than duplicating existing edges. An improved version of mk_even_graph could make use of the algorithm of Edmonds and Johnson (1973) which solves the Chinese postman problem.

5 Comparison of algorithms

In the previous sections we described four methods for constructing Eulerians. The first three, hpaths, weighted_hpaths and eseq construct Eulerians on K_n only, while the function eulerian is appropriate for connected graphs. Table 1 summarizes these properties.

Table 1 Properties of four algorithms

Algorithm	Graph	Hamiltonians	Weights
eseq	complete	no	no
hpaths	complete	yes	no
weighted_hpaths	complete	yes	yes
eulerian	connected	no	optional

For unweighted complete graphs with n vertices, PairViz offers three choices, hpaths(n), eseq(n) or eulerian(n). The algorithm hpaths produces Hamiltonians, so occurrences of nodes are spread out more or less uniformly throughout the path, as shown in Figure 4 for $n = 8$ and $n = 9$. Figure 8 shows the sequences produced by eseq and eulerian for $n = 8$ and $n = 9$. As explained in Section 4.3, eseq visits edges between low order nodes first, which might be advantageous if there were are a lot of nodes and the low order ones and visits between them are more important. The classical algorithm implemented in the function eulerian always visits the next available node (when the graph is unweighted), which is the lowest-index available node since the nodes are ordered $1,\ldots,n$. Essentially, eseq is based on an ordering of edges while eulerian for unweighted graphs is based on an ordering of nodes.

For even n, the results of these two strategies are identical. Compare, for example, the sequences eseq(8) and eulerian(8) shown in the first row of Figure 8. For odd n, the the eseq(n) and eulerian(n) sequences are different. The second row of Figure 8 shows the sequences eseq(9) and eulerian(9). The first part of the sequences until the path reaches node "7" are identical. Due to its recursive definition eseq(9) visits the edge (7,3) before the edge (7,1). In contrast eulerian(9) on its first visit to node "7" moves to the lowest numbered available node which is node "1", hence edge (7,1) is visited before (7,3).

For weighted complete graphs PairViz offers a choice between the functions weighted_hpaths and eulerian. As an example, consider the weighted complete graph based on the distances between 21 European cities given in eurodist (part of the standard R release). Figure 9 shows traces of the distances as ordered by Eulerians on K_{21} produced by the functions eseq, eulerian and weighted_hpaths. As expected the distances in the first trace show no particular pattern, while those in the second trace formed by eulerian have a strong increasing pattern. It is not so obvious that the distances in the third trace increase. This sequence is a Hamiltonian decomposition, with the 10 Hamiltonians separated by the dashed vertical grid in Figure 9. The 10 Hamiltonians are ordered by the total of their weights, which are inter-city distance in this application. The overlaid trace shows the average inter-city distance between successive cities in each of the 10 Hamiltonians, and this trace has a weak increasing trend. In this setting weighted_hpaths is not so successful, but given that eurodist consists of distances between 21 European cities, this is not so surprising.

Finally, a word about efficiency. Since K_n has $O(n^2)$ edges, any Eulerian algorithm on K_n must be at least $O(n^2)$. In fact all of the algorithms for constructing unweighted Eulerians are $O(n^2)$. For weighted_hpaths

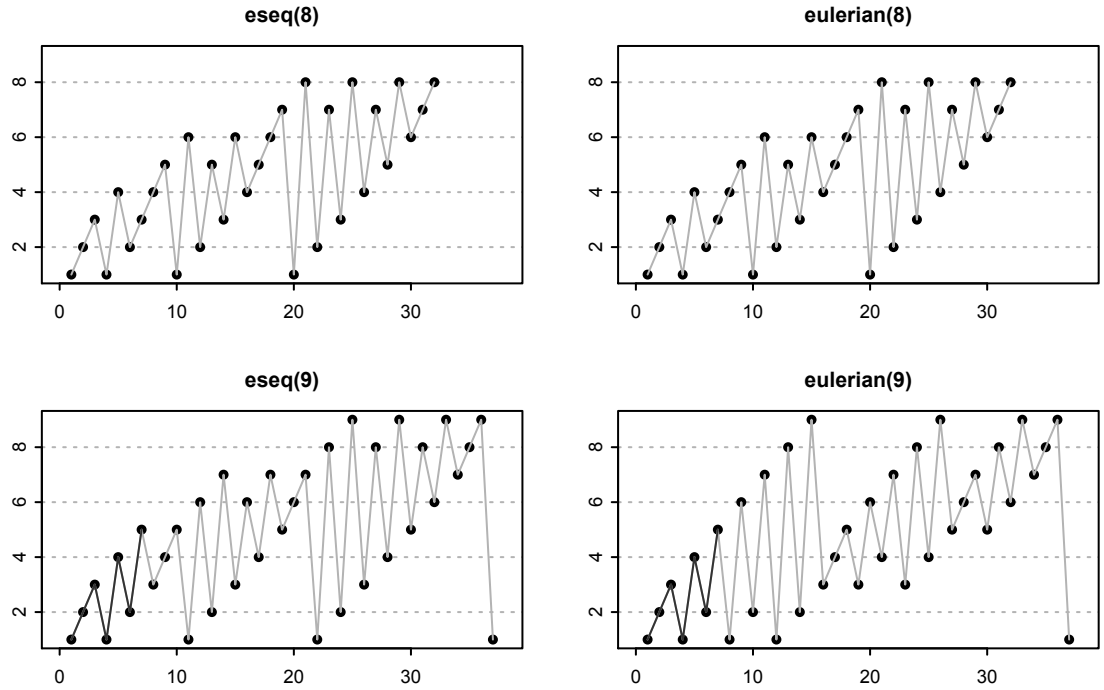


Fig. 8 Traces of the Eulerian sequences on K_8^e and K_9 generated by the functions `eseq` and `eulerian`. That is, the plots trace `eseq(8)`$[i]$, `eulerian(8)`$[i]$, `eseq(9)`$[i]$ and `eseq(9)`$[i]$ versus i, for $i = 1,2,\ldots,N$, where N is the sequence length.

finding the best Hamiltonian in the first step of the algorithm given in Section 4.2 is the bottleneck. With the `eulerian` function on weighted graphs, the cost associated with constructing an ordered Eulerian must include the cost of an edge sort at each vertex, and so has overall order on K_n of $O(n^2 \log n)$.

However these bounds are not the full story. In practice `eseq` is substantially faster than the `eulerian` function, even when weights are ignored. For example, on K_{60} `eulerian` takes about 20 seconds while `eseq` takes less than 1/100th of a second on a 2.66 GHz MacBook Pro. It would seem that for this application `graphNEL` may not be an efficient choice of data structure.

6 A model example

In this section we outline the use of Eulerians for regression model comparison.

We will use the well-known sleep data (Allison and Cicchetti 1976), with four predictors, non dreaming sleep (A), dreaming sleep (B), log body weight (C) and maximum life span (D) and a response of log brain weight, recorded for 45 mammal species. (Species with missing data are omitted). Figure 10 represents model choices by nodes of a graph. Edges connect nodes whose models differ by adding or removing one predictor. This graph is actually a *hypercube graph* Q_n with $n = 4$. Each node has an even number of edges (as is true for any Q_n with even n) so the graph is Eulerian.

Stepwise regression algorithms begin with the full model (or the best single predictor model), and repeatedly remove (or add) predictors. The edges in the graph of Figure 10 shows the set of possible moves in a stepwise regression algorithm. Edge weights (not shown) are given by the absolute difference in residual sums of squares of the models at its adjacent nodes (i.e, the extra sum of squares). Then we form an Eulerian starting with the full model and explore the residuals for each visited model (node) in Figure 11. With this display we can compare how the residuals change from one model to the next. At each stage, the weighted Eulerian algorithm moves to the most similar model where one variable has been added or dropped. Note that the sequence of models visited by the Eulerian does not correspond to that produced by any version (backwards, forwards or mixed) of a stepwise regression strategy.

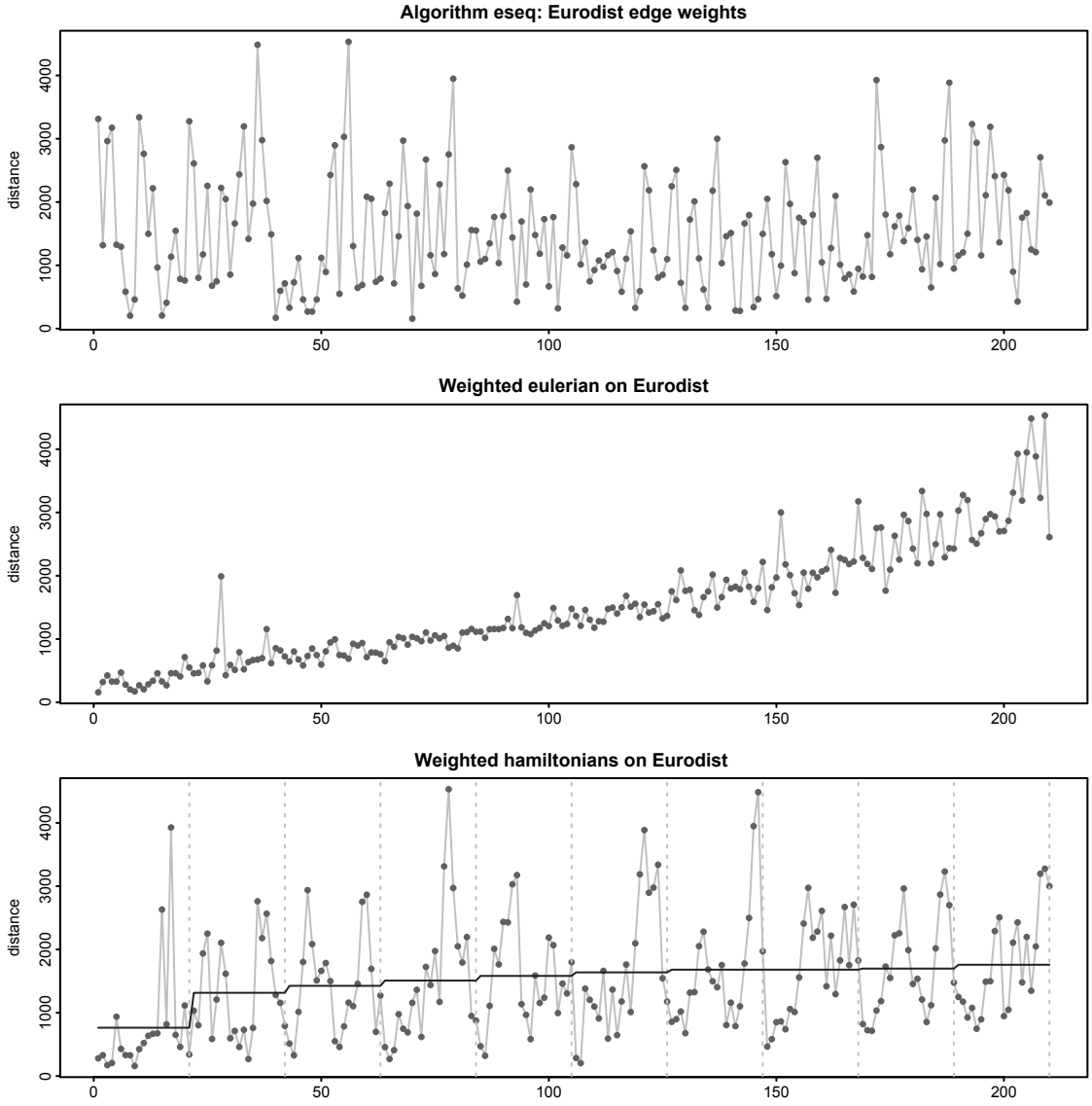


Fig. 9 The three panels show traces of the distances in the distance matrix `eurodist`. The top panel uses the path `eseq(21)`, the middle panel uses `eulerian(eurodist)`, and the bottom uses `weighted_hpaths(eurodist)`. In each case, for a path `p` the panel shows `eurodist[p[i],p[i+1]]` versus i, for $i = 1,2,\ldots,N-1$, where N is the length of the Eulerian `p` on K_{21} The bottom panel also shows the average distance in each of the 10 Hamiltonians (marked by the dashed vertical grid) of `weighted_hpaths`.

The accompanying barchart guide of Figure 11 shows the increase or decrease in residual sum of squares in moving from one model to the next. The (absolute) height of each bar gives the edge weights of the graph shown in Figure 10.

In Figure 11 we see that the residuals for the first 10 models visited appear to change very little. The first big increase in the residual sum of squares occurs when predictor C is dropped from model ABC. This is the first model visited which does not include predictor C (log body weight), and in fact all models with body weight as a predictor are included in the first 10 nodes visited. A revised plot which zooms in on the first 10 axes will explore these models in more detail, or alternatively, one could construct an Eulerian on the graph obtained by removing nodes representing models which do not include predictor C.

Note that the hypercube graph Q_4 has 32 edges, and so the Eulerian presented in Figure 11 has 33 axes and fits comfortably across an A4-sized page. If we add one extra predictor we have a graph Q_5 with $2^5 = 32$ nodes and 80 edges, but the graph requires the addition of extra edges to become Eulerian. In fact, the

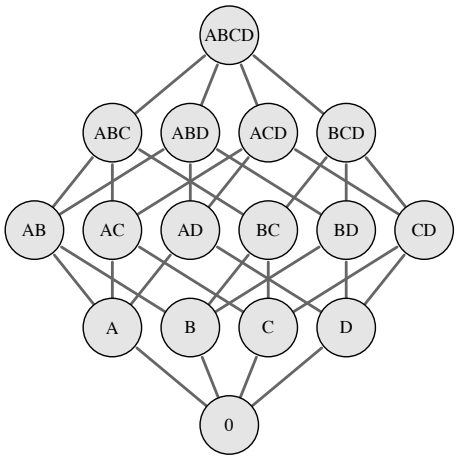


Fig. 10 In this graph, nodes represent models. Node '0' is the intercept-only model. Edges connect nodes when models differ by adding or removing one predictor.

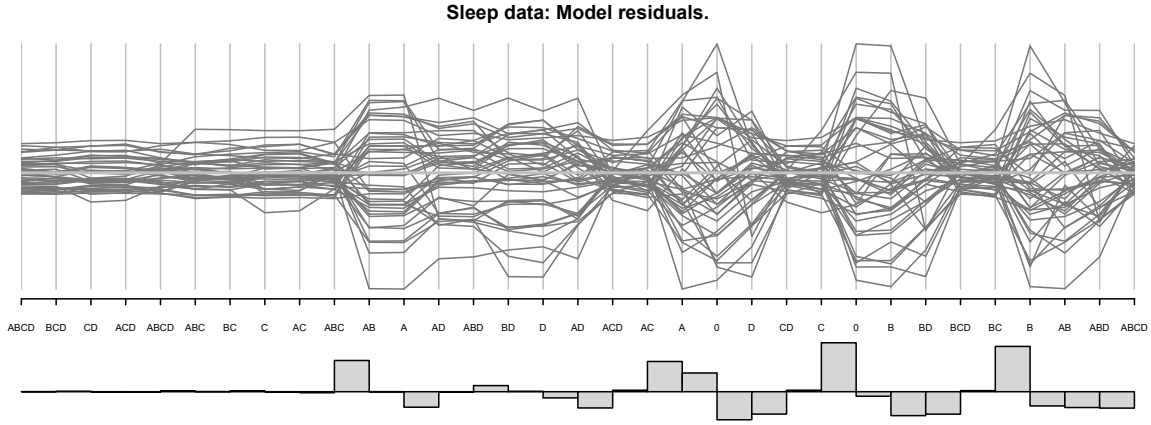


Fig. 11 Parallel coordinate display of model residuals. Barchart guide shows increase or decrease in residual sum of squares in moving from one model to the next.

Eulerian parallel coordinate plot has 97 axes, and would require two or three rows of an A4 page. Perhaps more usefully, we could reduce the graph size, removing nodes (and thus edges) by selecting say the two best models for each number of predictors, using some criterion such as mean squared error or Mallows' Cp.

Unwin et at (2003) demonstrate comprehensively the value of parallel coordinate displays in exploratory modeling analysis. They use axis orderings which visit all one-predictor models, then all two-predictor models and so on. Eulerian tours of model space are a useful addition to the model exploration toolkit. With them we can see precisely how model summaries change as predictors are added or dropped. In this example we chose residuals and their sum of squares as our model summaries, but other quantities could equally well be adopted.

7 Concluding remarks

The `PairViz` package (Hurley and Oldford 2009) is for pairwise visualization and comparison of statistical objects, using Hamiltonian decompositions and Eulerian paths. In this paper, we focused on the various algorithms for Hamiltonian decompositions and Eulerian paths.

However, this methodology leads to deeper insight of some conventional statistical graphics, and brings home to the data analyst how the choice of Hamiltonian in a visualization confounds the interpretation. The raters example in Section 2 illustrates this.

Our methodology leads to enhanced versions of conventional statistical displays, specifically, the parallel coordinate display with accompanying barchart guide, as shown in Figures 2 and 11. This display is available via the `guided_pcp` function in `PairViz`, while the enhanced version of the parallel coordinate display for categorical data is provided by the function `catpcp`. In Hurley and Oldford (2010a) we used `guided_pcps` to look at scagnostics (Wilkinson et al. 2005) of pairwise variable displays.

Shifting the focus of a visualization to showing the comparison between objects led us to a new graphical method for multiple pairwise comparison of treatment groups (Hurley and Oldford 2010a). This new visualization is available via the `mc_plot` function in `PairViz`.

The basic notion that graphs and graph traversals are a useful model for data visualization also suggests new ways of looking at dynamic graphics (Hurley and Oldford 2010b).

While in principle the algorithms presented here yield Eulerians on graphs of arbitrary size, screen or page size, as well as human attention capacity restricts the length of edge-traversal sequences that can be usefully presented. As hinted in Section 6, dimension reduction methods which reduce the graph size (the number of nodes and or number of edges) should prove useful here.

8 Appendix

Property 1 For odd n, `eseq-odd(n)` has length $E_n = \binom{n}{2} + 1$. The first E_k elements gives `eseq-odd(k)`, for $k = 1,3,5,\ldots,n-2$. For even n, `eseq-even(n)` has length $E_n = n^2/2$. The first E_k elements gives `eseq-even(k)`, for $k = 2,4,\ldots,n-2$.

Proof For odd n, `eseq-odd(n)` is by construction an Eulerian on the graph K_n. This graph has $\binom{n}{2}$ edges, all of which are visited exactly once by a sequence of length $\binom{n}{2}+1$. The second assertion follows from the recursive construction of `eseq-odd(n)` described in Section 4.3.

For even n, `eseq-even(n)` is by construction an Eulerian on the graph K_n^e. The graph K_n has $\binom{n}{2}$ edges, and K_n^e an additional $(n-2)/2$ edges all of which are visited exactly once by a sequence of length $\binom{n}{2} + (n-2)/2 + 1 = n^2/2$. The second assertion follows from the recursive construction of `eseq-even(n)` described in Section 4.3.

Property 2 Consider two nodes $i < j$ of the graph K_n, and an integer $k \geq 1$. For most values of i, j, k, the `eseq-odd` and `eseq-even` sequences vist the edge $(i-k, j)$ before (i, j) and the visit to (i, j) precedes a visit to $(i, j+k)$. The exceptions are given in the proof below.

Proof First, we consider the `eseq-odd` sequence.

To check when edge (i, j) is visited before $(i, j+k)$, we examine the j odd and even cases separately. When j is odd, the edge (i, j) is visited before $(i, j+k)$. This is because (i, j) is an edge in K_j and so occurs in `eseq-odd(j)`, while $(i, j+k)$ is not an edge in K_j and is visited later in the Eulerian.

When j is even, (i, j) occurs in `eseq-odd(j+1)`, while for $k \neq 1$, $(i, j+k)$ does not occur `eseq-odd(j+1)`, and so (i, j) is visited before $(i, j+k)$. Actually, checking the formulation for T_{new}, we can see that for $i = 1$ or even, (i, j) is visited before $(i, j+1)$.

We now check when edge $(i-k, j)$ is visited before (i, j). Except for the closing '1' at the end of T_{new}, the T_{new} sequence visits nodes $1, 2, \ldots, n-2$ in order, and so edge $(i-k, j)$ is visited before (i, j) unless $i-k = 1$ and j is odd. In this j odd case, edge $(1, j)$ is visited after edge (k, j) for $k = 2, 3, \ldots, j-1$.

In summary, the `eseq-odd` sequences vists the edge $(i-k, j)$ before (i, j) unless $i-k = 1$ and j is odd, and the visit to (i, j) precedes a visit to $(i, j+k)$ unless j is even, $k = 1$ and $i \neq 1$ is odd.

Next, we consider the `eseq-even` sequence.

As before, to check when edge (i, j) is visited before $(i, j+k)$, we examine the j odd and even cases separately. When j is even, the edge (i, j) is visited before $(i, j+k)$. This is because (i, j) is an edge in K_j and so occurs in `eseq-even(j)`, while $(i, j+k)$ is not an edge in K_j and is visited later in the Eulerian.

When j is odd, (i, j) occurs in `eseq-even(j+1)`, while for $k \neq 1$, $(i, j+k)$ does not occur in `eseq-even(j+1)`, and so (i, j) is visited before $(i, j+k)$. Furthermore, from the construction of T_{new}, we can see that for odd i only, (i, j) is visited before $(i, j+1)$.

We now check when edge $(i-k,j)$ is visited before (i,j). By inspecting the T_{new} sequence which visits nodes $1,2,\ldots,n-2$ in order, we see that $(i-k,j)$ is visited before (i,j). However, for odd j, edge $(j-1,j)$ is visited twice. The first visit is the joining edge between `eseq-even(j-1)` and `eseq-even(j+1)` and this visit occurs before visits to (k,j), for $k=1,2,\ldots,j-2$.

In summary, the `eseq-even` sequences vists the edge $(i-k,j)$ before (i,j) unless $i=j-1$ and j is odd, and the visit to (i,j) precedes a visit to $(i,j+k)$ unless k = 1, i is even and j is odd.

References

Allison T, Cicchetti D (1976) Sleep in Mammals: Ecological and Constitutional Correlates. Science 194:732-734.

Alspach B, Bermond JC, Sotteau D (1990) Decomposition into cycles I: Hamilton decompositions. In: Hahn G, Sabidussi G, Woodrow R.E. (eds) Cycles and Rays, Kluwer Academic Publishers, Boston, pp 9-18.

Ankerst M, Berchtold S, Keim DA (1998) Similarity Clustering of Dimensions for an Enhanced Visualization of Multidimensional Data. IEEE Symposium on Information Visualization 1998:52-60.

Bendix F, Kosara R, Hauser H (2005) Parallel Sets: Visual Analysis of Categorical Data. IEEE Symposium on Information Visualization 2005:1-18.

Cleveland WS (1995) Visualizing Data. Hobart Press, Summit, NJ

Csardi G, Nepusz T (2006) The igraph software package for complex network research. InterJournal, Complex Systems 1695.

Edmonds J, Johnson EL (1973) Matching, Euler Tours, and the Chinese Postman. Mathematical Programming 5:88-124

Fleiss JL(1971) Measuring nominal scale agreement among many raters. Psychological Bulletin 76, 378-382.

Friendly M, Kwan E (2003) Effect Ordering for Data Displays. Computational Statistics and Data Analysis, 43, 509-539.

Gamer M, Lemon J, Fellows I (2009) irr: Various Coefficients of Interrater Reliability and Agreement. R package version 0.82.

Gentleman R, Whalen E, Huber W, Falcon S (2010) graph: A package to handle graph data structures. R package version 1.26.0.

Gross JL, Yellen J (eds.) (2004) Handbook of Graph Theory, CRC Press, London.

Hierholzer C. (1873) Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. Math. Annalen VI:30-32.

Hofmann H. (2006) Multivariate categorical data–mosaic plots. In: Unwin A, Theus M, Hofmann H (eds) Graphics of large datasets, visualizing a million, Springer, New York, pp 105-124.

Hurley CB (2004) Clustering Visualizations of Multidimensional Data. Journal of Computational and Graphical Statistics, 13:788-806.

Hurley CB, Oldford RW (2009) PairViz: Visualization using Eulerian tours and Hamiltonian decompositions, R package version 1.1.

Hurley CB, Oldford RW (2010a) Pairwise display of high dimensional information via Eulerian tours and Hamiltonian decompositions. Journal of Computational and Graphical Statistics, to appear.

Hurley CB, Oldford RW (2010b) Graphs as navigational infrastructure for high dimensional data spaces. Computational Statistics, this issue, please update DOI.

Lucas DE (1892) Recréations Mathématiques, Vol. II. Gauthier Villars, Paris.

Unwin A, Volinsky C, Winkler S (2003) Parallel coordinates for exploratory modelling analysis. Computational Statistics and Data Analysis, 43:553-564 .

Wegman EJ (1990) Hyperdimensional data analysis using parallel coordinates. Journal of the American Statistical Association, 85:664-675.

Wilkinson L, Anand A, Grossman R (2005) Graph-theoretic scagnostics. IEEE Symposium on Information Visualization 2005:157-164.

Theus M (2002) Interactive Data Visualization using Mondrian. Journal of Statistical Software 7(11)1-9.

Wills G (2000) A good, simple axis. Statistical computing and statistical graphics newsletter 11:20-25.

Yang J, Peng W, Ward MO, Rundenmeister EA (2003) Interactive hierarchical dimension ordering, spacing and filtering for exploration of high dimensional datasets. IEEE Symposium on Information Visualization 2003: 105-112.